
Ritrovato, P.; Xhafa, F.; Giordano, A. Edge and cluster computing as enabling infrastructure for Internet of Medical Things. *32nd IEEE International Conference on Advanced Information Networking and Applications, IEEE AINA 2018, 16-18 May 2018, Krakow, Poland: proceedings*, p.717-723. Doi: 10.1109/AINA.2018.00108

Ritrovato, P.; Xhafa, F.; Giordano, A. Edge and cluster computing as enabling infrastructure for Internet of Medical Things. *32nd IEEE International Conference on Advanced Information Networking and Applications, IEEE AINA 2018, 16-18 May 2018, Krakow, Poland: proceedings*, p.717-723. Doi: 10.1109/AINA.2018.00108

Edge and Cluster Computing as Enabling Infrastructure for Internet of Medical Things

Pierluigi Ritrovato
DIEM
Università degli studi di Salerno
Salerno, Italy
Email: pritrovato@unisa.it

Fatos Xhafa
Departament de Ciències de la Computació
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: fatos@cs.upc.edu

Andrea Giordano
DIEM
Università degli studi di Salerno
Salerno, Italy
Email: andrea.giordano.inf@gmail.com

***Abstract*—The continuous adoption of fitness and medical smart sensors are boosting the development of Internet of Medical Things (IoMT), reshaping and revolutionizing Healthcare. This digital transformation is paving the way to new forms of care based on real-time analysis of huge amounts of data produced by sensors, which is seen as a basis for improving clinical efficiency and helping to save lives. A medical sensor typically produces several KBs of data per second so the collection and analysis of these data can be approached with Big Data technologies. The aim of this paper is to present and evaluate a hybrid architecture for real-time anomaly detection from data streams coming from sensors attached to patients. The architecture includes an edge computing data staging platform based on Raspberry Pi 3 for data logging, data transformation in RDF triple and data streaming towards a cluster computing running Apache Kafka for collecting RDFStreams, Apache Flink for running a parallel version of the Hierarchical Temporal Memory algorithm and Cassandra for data storing. The different layers of the architecture have been evaluated in terms of both CPU performance and memory usage using the REALDISP dataset.**

***Index Terms*—eHealth, IoMT, wearables, Big Data, stream computing**

I. Introduction

The continuous progress in the development of powerful embedded computing systems together with their reduced production costs are contributing to the fast development of products and solutions in many fields. Healthcare is among the most prominent one due to the potential impact from organizational, economical and social point of views. Today, the most developed countries spend at least 10% of their GDP in Healthcare [2], indeed, information technologies, together with an improved quality for treatments, lead to a reduction of expenses avoiding hospitalisation for those patients whose diseases can be remotely screened. In this sense, the application of innovative technologies represents a far-sighted investment to enhance both medical and financial services.

In World Healthcare Organisation 2017 statistics report [3] is revealed that severe cardiac diseases like heart stroke and ischaemic heart disease are the most frequent causes of death in the world. These events can eventually be alleviated and even avoided with an appropriate use of current available technology.

In this paper we propose a system architecture to perform real-time detection of anomalous pattern in data streams collected from wearable sensors. To define "wearable" concept, we can consider the following: "*wearable sensors are devices that can be worn or mated with human skin to continuously and closely monitor an individual's activity, without interrupting or limiting the user's motion*" [4]. Vital signs measurement and motion tracking, like accelerometers and gyroscopes, are among the most used sensors in eHealth: the latter are useful in fall risk assessment and statistical study on patient's habits, whereas the former are used to detect ECG, EEG, skin temperature, etc. . The explosion in the development and adoption of smart medical sensors for healthcare has lead to the definition of the so called Internet of Medical Things (IoMT), which are revolutionizing the Healthcare worldwide. In 2015 69% of US citizens track their health with various sensors; in UK, 7 in 10 adults tracks their health as reported in [5] with a double digit growing trend. Indeed, according to Gartner[1], 310M of devices has been sold in 2017.

From the computer science and engineering point of view Healthcare is an impressive producer of data: even in a small clinic there could be hundreds of patients and for each one, several vital parameters have to be monitored. Assuming a few and continuously active sensors for several days, leads us to confirm that Healthcare data are characterized by the 4 V's (*Velocity*, *Volume*, *Variety* and *Veracity*) motivating us to looking at Big Data technologies to analyze these data.

Healthcare can take advantages also from the use of Semantic Web technologies: semantic data analysis adds meanings to raw, and sometimes apparently useless, data pulled out from sensors that would be otherwise discarded and unused. Then, Semantic Web technologies allows to multiply the already huge amount of knowledge extracted from the fetched data in both streaming and batch way.

The remainder of the paper is organized as follows. The Section II introduces some of most relevant works in eHealth, exposing their key points. A layered architecture is described and each block is deepened in details in Section III. The Section IV presents results coming from an evaluation of the system and, finally, the Section V contains some final considerations on the proposed architecture and indicates future

[1]https://techcrunch.com/2017/08/24/global-wearables-market-to-grow-17-in-2017-310m-devices-sold-30-5bn-revenue-gartner/

works.

II. Related works

With the growing relevance of the eHealth, a large number of researchers have reported systems and solutions to analyse physiological data coming from wearable sensors attached to patients; some of them have considered Big Data streams, as we do in this paper, but to date pattern detection or real-time processing has not been deeply explored. For instance, the system described in [7] remotely monitor patients using data from ECG and accelerometers: the application reports to clinicians periods of elevated heart rate and filters expected critical situations, during a run, for instance.

Many researchers agree to identify powerful micro-controllers and embedded systems as fundamental components in any eHealth system for collecting data from wearables. The most preferred ones are *Raspberry Pi* and *Arduino* due to their computing capability, reduced costs and diffusion. Moreover, a large part of researchers have focused on the analysis of data related to a single individual instead of considering a system able to handle potentially hundreds of persons. An example is presented in [8] that describes a single board computer, which measures ECG of a specific patient and stores data on an SD card in order to analyse it later. The authors in [9] instead have implemented a sensor network to monitor patient's heart rate based on a domestic monitoring system. Finally, [10] presents a system to record physiological data of human body.

III. Architecture

The proposed system in this paper follows a different approach with regard to the works presented in Section II. There are some reasons which lead us to choose a different way to deal with the proposed problem, which we briefly discuss in the following. The majority of existent works in the literature are focused on single patient whereas a global approach to take into account medical structures should be preferred according to the IoMT concept. Indeed, a more comprehensive approach could be more effective since it would give the opportunity to treat not only the single patient but also to improve the Healthcare research as a whole, for instance by creating statistics about efficacy of drugs and treatments. Secondly, the proliferation of several personal care systems does not allow to establish a standard way to manage health data and could prevents the Healthcare digitalization process.

Our solution makes use of a mixed architecture composed of embedded devices and cluster infrastructures in order to deal with the large amount of data streams produced in a real medical scenario, to reduce data loss and to guarantee continuous availability required by critical applications. The use of a single machine presents many difficulties to handle such a complex task, making the system not reliable and viable in real context; furthermore, it would limit the opportunities for future re-factoring and extensions of the system.

In order to satisfy these requirements our solution exploits a cluster infrastructure. The system is not thought for personal use whereas it fits the requirements of hospitals, clinics and medical organizations, which want to improve effectiveness of treatments and diagnosis. Our solution acts as a technologically forefront instrument to help clinicians to obtain more meaningful data in shorter times. The proposed architecture thus allows to continuously fetch data coming from a number of wearable sensors attached to patients and analyses the resulting data streams. The aim is to detect potential anomalies in real-time in order to predict and identify emergencies.

Some of the most important system's requirements are the following:

- *Bearing a large number of sensors which send data concurrently.*
- *All the analyzed data must be stored in a persistent storage together with detected anomalies.*
- *Data must be replicated through the cluster.*
- *Ensure proper reaction in case of hardware failure with minimal data loss.*
- *Almost real-time analysis of data generated by wearable sensors.*

The architecture of the system is composed of 4 layers as depicted in Fig.:1:

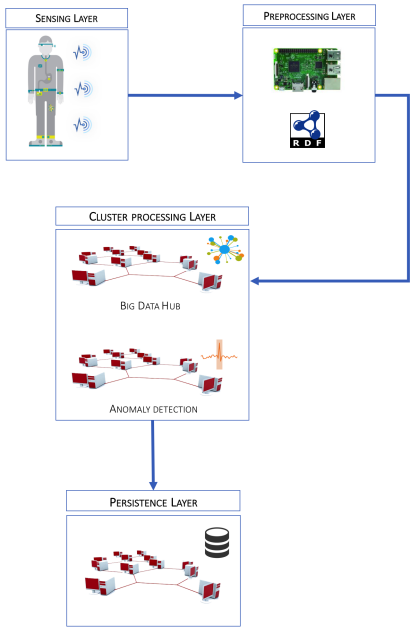

Fig. 1. Schematized system's functioning.

A. Sensing layer

The first block of the architectural stack, named *Sensing layer*, feeds the entire system. It is the patient-closest layer and is represented by health wearables: depending on the specific sensor it can use several technologies (for instance wired or wireless connection) to fetch and dispatch data.

B. Pre-processing layer

The *Pre-processing layer* leverage on an embedded board, which retrieves data produced by wearables. The main goal of

this block is to collect, as an unbounded data stream, inputs coming from the *Sensing layer* and to convert raw sensor data into RDFStreams. Finally, the converted streams are sent to the cluster infrastructure for further processing.

Raspberry Pi 3 was chosen as embedded board due to its computing capability and cost. It hosts a Linux OS and a *Node-Red* server, a programming tool which offers a number of virtual computing nodes to simplify program implementation. In order to get data coming from the *Sensing layer* the facilities offered by *Mosquitto* nodes are exploited. The use of the MQTT broker allows to maintain a technology independence in term of communication between wearables and the board; furthermore, it provides also a temporarily memory to retain and to retrieve generated data in case of network lacks or if a failure of the subsequent layers occurs.

An example of the designed *Node-Red* programming flow is depicted in Fig.:2. The first node represents a *Mosquitto* consumer: its responsibility is to fetch data from the previous layer; then, the received data are represented with a simple JSON format and sent out towards the other nodes. The second and third nodes are responsible to parse and convert the JSON message into a JSON-LD, which is the standard used to represent RDFStreams. This fundamental step adopts W3C recommendations to express semantic data streams following an approach inspired by the TripleWave framework [11] for data conversion. The last part of the depicted pipeline dispatches converted data to the next layer using a revised version of the *node-red-contrib-kafka-node* [12].

Fig. 2. Example of a Node-Red programming flow.

C. Cluster processing layer

The *Cluster processing layer* is composed of two clusters of commodity hardware; it must collect semantic-enriched data and detect anomalous pattern within data streams. We can separate the tasks of the two clusters: The first one represents the "hub" of the application, it must buffer data and be a safe dock to store temporarily messages which have to be analyzed. The second one must run in real-time an anomalies detection algorithm and reveals critical situations or emergencies.

We have chosen Apache Kafka to perform the task deployed on the first cluster because it is one of the most important Big Data Messaging Hub. It is scalable, fault tolerant and guarantees high performance [13]. The Kafka cluster is composed of 3 brokers: this number is considered an appropriate replication factor (from here RF) [13]. Brokers are identical and each one hosts a number of Kafka topics equal to the number of sensors' types available in the system: in this way all data belonging to a specific sensor falls in the same topic and every consumers can retrieve the right data just specifying the requested sensor. In the system, Kafka acts as a bridge between the *Preprocessing layer* and the processing cluster and offers an access point for external systems which want to consume semantic data.

The second cluster performs a stream processing of semantic-enriched data coming from *Preprocessing layer* and collected by the Kafka cluster. Current Big Data panorama presents many distributed stream processors, which differ in performance and characteristics. Three stream processors were considered for this work: Apache Spark, Apache Storm and Apache Flink. Unlike Storm, the others are both batch and stream processors. Actually, Spark utilizes a technique called "micro-batching" which consists of splitting data streams into small chunks and performing a quick batch processing on them. It should be noted that this approach does not allow to reach optimal performance. On the other hand, Storm and Flink provide true real-time processing and achieve both great results in terms of low latencies and high throughputs. Since the latter offers higher-level API and presents better results in some benchmarks as reported in [14] [15], Apache Flink has been chosen to fulfill the task of the second cluster. A brief comparison between these Big Data processors is depicted in TABLE I.

TABLE I
COMPARISON OF MAIN DISTRIBUTED BIG DATA STREAMING PROCESSOR.

	SPARK	**STORM**	**FLINK**
Batch processing	✓		✓
Stream processing	Micro batching	✓	✓
Latency	Medium	Very low	Low
Throughput	High	Low	Medium
Fault tolerant	✓	✓	✓
Kafka supporting	✓	✓	✓

The purpose of the Flink cluster is to get data from Kafka cluster and process them: in particular, Flink parses data streams stored in Kafka and detects potential anomalies within them. In order to do it, a distributed version of Hierarchical Temporal Memory (HTM) algorithm [16] has been implemented on Flink. The HTM distributed implementation is detailed later in the Section III-E. In Flink the time dimension can be handled in many ways but for our purpose, since it is particularly important for data analysis, a logic based on the timestamps associated to each physiological value was adopted: Flink names it *Event time* logic. Medical values are strongly dependent by the instant they are generated and the anomaly detection algorithm strictly processes data in chronological order. In order to guarantee that HTM processes data correctly, Flink sorts messages as they arrive looking at the generation timestamp. Sometimes messages can be delivered to Flink with a certain delay, due to network lacks or for processing latency: if the delay is lower than a fixed threshold, Flink re-insert the specific message in the stream correctly, otherwise the message is discarded. On one hand, this approach introduce some delay because it is not implemented natively by Flink (to date), on the other hand it allows to execute an accurate anomaly detection and to lose just a

minimal number of out-of-order messages.

D. Persistence layer

The *Persistence layer* is deputy to store data analyzed by the *Cluster processing layer* in order to allow further elaborations in the future. The layer provides also an access point for external systems to retrieve stored data.

TABLE II illustrates a comparison between the frameworks considered to design the *Persistence layer*: Apache Cassandra and Apache HBase. Both are NoSql databases able to provide distributed storages and both belong to the family of column-oriented databases, but they differ in performance and offered facilities. Cassandra exposes a query language, called CQL, which is very similar to SQL so could facilitate migrations for those who are interested in developing advanced reporting functionalities. Moreover, Cassandra provides a greater flexibility than HBase in terms of consistency control [17]. According to the benchmark exposed in [18] Cassandra beats HBase for number of operations executed per second in load process context. This effect is highlighted in a scenario with a balanced number of writes and reads and Cassandra overtakes HBase hundreds times [19]. Moreover, we have to consider that the *Persistence layer* mainly executes write operation and Cassandra is optimized for this kind of operation as demonstrated in [20]. For these reasons we decided to implement the *Persistence layer* using the Cassandra framework.

TABLE II
COMPARISON BETWEEN APACHE CASSANDRA AND APACHE HBASE.

	CASSANDRA	**HBASE**
No single points-of-failure	✓	
Supported languages	13	8
Optimized operation	Write	Read
API	Proprietary	Java
Concurrency	✓	✓
Durability	✓	✓

Apache Cassandra offers a P2P architecture which avoids single point of failure and allows to reach high availability more easily with respect to other storage systems: in the cluster there are no masters and the cluster topology is more similar to a ring with identical nodes. A RF of 3 is considered enough in most cases according to [21]. The data replication strategy to set depends on the specific scenario in order to guarantee high-availability and to avoid data loss. Thinking about an installation of our system in a single hospital, a single rack of nodes could be enough as well as a simple data replication through the ring; on the other hand, if the system must handle a group of clinics or a medical institution which owns several hospitals, probably a cluster geographically spread in multiple racks is more adequate and the *Network Topology Strategy* is more appropriate because it replicates data across every racks.

In order to provide best performance in reading data, NoSQL tables must be defined on expected queries [21]. Since that, we specified the following queries:

1) *Retrieve every value belonging to a specified sensor of a particular patient;*
2) *Retrieve every abnormal value belonging to a specified sensor of a particular patient;*
3) *Retrieve every value belonging to a specified sensor of a particular patient in a fixed time interval;*

These queries lead to the creation of two specific tables. The first table contains all detected values and specifies the code of the patient, the name of the specific sensor that detected the value, the observed parameter, the timestamp, the value read and a verbose representation of the related RDFStream. The second table contains just the values which are identified as anomalous: each record is composed of the same fields included in the previous table, plus a field that indicates the anomaly index of the specific value. The *compound partition key* for both tables is the pair patient-sensor while the *clustering key* is the timestamp.

E. Hierarchical Temporal Memory algorithm

In Healthcare, discovery of anomalous vital signs has an extreme importance to prevent sudden diseases and to assure an immediate medical intervention in order to solve risky situations as soon as possible. Generally, an anomaly is defined as a point in time where the behaviour of the system is unusual and significantly different from the past [22]. This definition implies that an anomaly can be considered, in general, as an unusual value in a continuous data flow (stream). HTM is a foundational technology for the future of machine intelligence. It is inspired to the biology function of the neocortex and provides some algorithms based on continuous unsupervised learning so it does not need a training step on data [23]. Moreover, it can be applied on almost every kind of data.

Roughly, as illustrated in Fig. 3, the HTM algorithm, starting from the current input *X(t)*, computes an input sparse binary code representation *a(Xt)* and a prediction of the *a* value for the next input. These values contribute to calculate *S(t)*: a raw anomaly score constrained between 0 and 1 and computed each time new values arrives. In general, if a *S(t)* is close to 1 means the correspondent *X(t)* is considered more anomalous than another closer to 0. More details and use cases of HTM algorithms are described in [16].

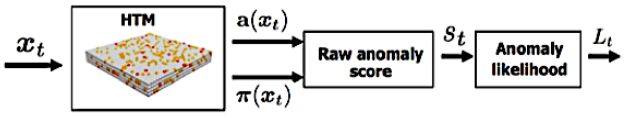

Fig. 3. HTM anomaly detection algorithm schema [16].

In order to use HTM on the Flink cluster, a distributed version the algorithm, called Flink-HTM [24] was used. The first step to execute the anomaly detection algorithm is to build a network. Creating it from scratch could be a very complex task due to the number of parameters to set and because it requires a strong knowledge of neural networks and machine learning topics. The construction of a network is strongly

related to the data it has to analyze. Actually the network adopted in this paper was built following a standard template which is considered generally adequate to the 90% of cases by the Numenta engineers. The network template was refined and improved in accordance with HTM documentations and community tips in order to make it more adherent to the provided data patterns. One of the most important tuned parameters is the network resolution: lower values produce a more accurate output but request more computational time. We can intend the resolution as a measurement of how much the values differs each other: if they are very close a more fine-grained resolution will be required to detect anomalies, otherwise just few anomalies will be found. Another important parameter to set is the threshold to distinguish anomalies from regular values: it has to be specified over the dataset as well and its definition requires some attempts; a too elevated threshold does not allow to reveal all anomalies while a too lower one would return many false positives.

IV. Experimental study

We run several tests to find out critical issues of the proposed architecture and to appraise the performance of the chosen frameworks evaluated in terms of both CPU and memory node usages. Other analyzed parameters are the average percentage of data loss and the processing latency. All these outcomes were obtained using Linux *sysstat* utility. About the data loss an average percentage of 0.3% was detected which in this scenario is considered an acceptable result since the loss of so few messages in a whole stream is not critical.

A. Adopted infrastructure

The hardware used for experimentations and tests was provided by *Research and Development Laboratory (RDLab)* [25] of Computer Science Department at UPC BarcelonaTech. The physical machines are based on Intel(R) Xeon(R) CPU X5550 2.66GHz octa-core. Each node hosts an instance of Ubuntu 12.04.2 LTS. The infrastructure is composed of 5 nodes: a node with a single-core CPU and 4 GB of RAM hosts a Kafka broker, a cluster of 3 identical nodes with dual-core CPU and 4 GB of RAM runs a Flink instance and finally a node with a dual-core CPU and 2 GB of RAM hosts a Cassandra broker. Finally, to implement the *Preprocessing layer* a Raspberry Pi 3 is used.

B. Dataset

In order to provide a realistic test we have used a fitness dataset, called *REALDISP* [1], to feed the system. It contains several log files of wearable sensors placed on different parts of 17 individuals where values are produced at 50 Hz rate: the whole dataset contains about 7 GB of data. Each record contains information about seconds and microseconds registered when data were collected. The wearable sensors set is composed of accelerometers, gyroscopes and magnetometers.

C. Sensing layer

In order to simulate the presence of real sensors, some scripts scan the dataset and send values to the next layer with a frequency of 50 Hz. Thus, data coming from sensors are forwarded to the next layer using *Mosquitto* producers. The output throughput of this layer for each sensor is around 6 KB/s or alternatively 50 msg/s.

D. Preprocessing Layer

The Raspberry Pi 3 involved in this layer retrieves data from the *Sensing layer* using 8 pipelines depicted in Fig. 2: each one runs independently and performs a stream enrichment. Since in this layer the received messages are semantically annotated the output throughput reaches 50 KB/s for each sensor. We tested the Raspberry Pi 3 with a raising number of active sensors, starting from 2 and reaching 32 streams: the results has been very good for both CPU and memory usages but as we added more than 32 sensors the embedded board began to lose messages (see TABLEIII).

TABLE III
Raspberry Pi 3 test outcomes.

N. FLOWS	**CPU** %	**MEMORY** %
2	33.29	62.73
4	33.69	64.26
6	41.09	63.91
8	58.02	56.21
10	66.53	62.73
12	69.81	61.11
14	72.88	65.65
16	78.50	70.46
32	80.43	70.11

E. Cluster processing layer

We evaluated the performance of Apache Kafka and Apache Flink separately. From the results shown in TABLEIV Kafka seems to be largely able to bear the provided throughput whereas it gets all available memory without worrying about the amount it really needs. Due to the Raspberry limits we could not test Kafka with more than 32 sensor flows.

TABLE IV
Apache Kafka test outcomes.

N. FLOWS	**CPU** %	**MEMORY** %
2	4,12	96.61
4	4.72	95.66
6	5.99	93.49
8	7.97	92.91
10	10.01	92.59
12	12.30	96.44
14	12.89	96.76
16	15.01	96.61
32	21.03	92.45

Evaluating Apache Flink, and so *Flink-HTM* library, was the most demanding part of the experimentation. Indeed, the

anomaly detection algorithm appears to be too heavy for the adopted infrastructure: with 2 sensor flows only, Flink was not able to complete the task in real-time unless we decrease the data delivery frequency from 50 Hz to 15 Hz. This outcome suggests that *Flink-HTM* library needs a more powerful infrastructure or alternatively a larger cluster. Like Kafka, also Flink presents the same greedy behavior about memory management. About CPU, with a delivery frequency of 15 Hz and 2 active flows, every CPU usage touch 100%.

F. Persistence layer

The evaluation of Apache Cassandra, since it is placed downstream to Flink, was analyzed separately. The layer was tested by developing a simplified Flink program which performs a dummy anomaly detection: with this expedient we were able to test Cassandra up to 64 sensor flows. Cassandra showed great performance about CPU usage and a greedy behaviour in memory usage (see TABLEV).

TABLE V
APACHE CASSANDRA TEST OUTCOMES.

N. FLOWS	**CPU** %	**MEMORY** %
2	4.27	81.42
4	8.96	84.40
6	13.21	88.98
8	14.37	95.83
10	17.00	91.6
12	17.50	93.67
14	18.53	96.78
16	18.90	96.76
32	22.37	95.83
64	33.33	93.95

V. CONCLUSION

In this paper we have presented a technology platform for real-time Healthcare data analytics. The aim is to use data streams from wearable sensors attached to patients to raise alarms or trigger autonomous reactions within few seconds about an emergency. The experimental evaluation considered several technological platforms. Besides Flink, in experimentation part we noted that all selected frameworks fulfilled the assigned task showing great performance. Concerning Flink performance they are mainly due to the intensive task represented by anomaly detection algorithm based on *Flink-HTM* library and seem to not be due to the framework *per se*. Although the designed architecture provides a reasonably well-working system, in order to implement a real-time anomaly detection in a real scenario a more powerful physical infrastructure is required. One of the most important strengths of the architecture is its modularity: it allows to easily expand and re-factor layers independently, replacing frameworks or adding new nodes to the processing clusters without changing the others elements of the infrastructure. Today, Healthcare is also a business opportunity: a good-working hospital guarantees patients to have good health but also great incomes for private companies and public entities. For these reasons each innovation is welcomed if it brings improvements and is not too expensive. In this regard, the presence of a modular architecture allows to size up the system and fit it on specific needs, avoiding waste of money and increasing the feasibility of the system.

REFERENCES

[1] Oresti Baños, Máté Attila Tóth, *Realistic sensor displacement benchmark dataset*, Dataset manual, 2014

[2] World Bank Group IBRD-IDA, Global Health Expenditure database, 2017 *https://data.worldbank.org/indicator/SH.XPD.TOTL.ZS* (accessed as of January 2018)

[3] World Health Organization Statistics, 2017, *http://www.who.int/mediacentre/factsheets/fs310/en/* (accessed as of January 2018)

[4] Wei Gao, Sam Emaminejad, Hnin Yin Nyein, Samyuktha Challa, Kevin Chen, Austin Peck et al. *Fully integrated wearable sensor arrays for multiplexed in situ perspiration analysis*, Nature, 2016; 529(7587): 509-14

[5] Ben Walker, *Every day Big Data statistics*, 2015 *http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/* (accessed as of January 2018)

[6] Tim Conrad, Lydia Ickler, Alexander Reinefeld, Florian Schintke, Robert Schmidtke, Christof Schtte et al. *Big Data Analytics in e-Health using Apache Flink and XtreemFS*, Berlin Big Data center, Zuse Institute Berlin, 2017

[7] Chung-Min Chen, Hira Agrawal, Munir Cochinwala, David Rosenblut, *Stream query processing for Healthcare bio-sensor applications*,20th International Conference on Data Engineering, 2004, IEEE

[8] Onder Yakut, Serdar Solak, Emine Dogru Bolat *Measuring ECG Signal Using e-Health Sensor Platform*, International Conference on Chemistry, Biomedical and Environment Engineering (ICCBEE'14), 2014

[9] Magaña-Espinoza P., Aquino-Santos R., Cárdenas-Benitez N., Aguilar-Velasco J., Buenrostro-Segura C., Edwards-Block A. et al. *WiSPH: A Wireless Sensor Network-based Home Care Monitoring System*, Sensors, 2014;14(4): 7096-7119

[10] Ioan Orha, Stefan Oniga, *Automated system for evaluating health status*, Design and technology in Electronic Packaging (SIITME), 2013, IEEE 19th International Symposium for, pp.219-222

[11] Andrea Mauri, Jean-Paul Calbimonte, Daniele Dell?Aglio, Marco Balduini, Marco Brambilla, Emanuele Della Valle et al. *TripleWave: Spreading RDF Streams on the Web*, International Semantic Web Conference, ISWC 2016: The Semantic Web - ISWC 2016, 2016, pp. 140-149

[12] F. Wang, 2016 *https://www.npmjs.com/package/node-red-contrib-kafka-node* (accessed as of January 2018)

[13] Kafka Official Documentation, 2017 *https://kafka.apache.org* (accessed as of January 2018)

[14] Ellen Friedman, Kostas Tzoumas, *Introduction to Apache Flink*, 2016, OReilly Media

[15] Reza Farivar, Kyle Knusbaum, *Performance Comparison of Streaming Big Data Platforms*, DataWorks Summit/Hadoop Summit, 2016

[16] Subutai Ahmad, Scott Purdy, *Real-Time Anomaly Detection for Streaming Analytics*, 2016, arXiv

[17] Rick Grehan, *Big data showdown: Cassandra vs. HBase*, 2014, InfoWorld *https://www.infoworld.com/article/2610656/database/big-data-showdown–cassandra-vs–hbase.html* (accessed as of January 2018)

[18] Datastax, *Benchmarking top NoSQL Databases*, 2015, End Point

[19] Birendra Kumar Sahu, *A real comparison of NoSQL databases*, 2015 *https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/* (accessed as of January 2018)

[20] Bogza A.G Adriana-Maria, *Performance evaluation of Apache Mahout for mining large datasets*, Master Thesis, FIB, UPC 2016. Under the supervision of Prof. Fatos Xhafa

[21] DataStax, *Apache Cassandra 3.0 Datastax documentation*, 2017, *https://docs.datastax.com/en/cassandra/3.0/* (accessed as of January 2018)

[22] Andrew Meola, *Internet of Things in Healthcare: Information technology in health*, Business Insider, 2016

[23] Numenta Community, 2017, Introduction to HTM, *https://numenta.org* (accessed as of January 2018)

[24] Eron Wright, flink-htm GitHub page, 2016 *https://github.com/htm-community/flink-htm* (accessed as of January 2018)
[25] Research and Development Laboratory, Universitat Politécnica de Catalunya, Facultat de Informatica de Barcelona, *https://rdlab.cs.upc.edu/* (accessed as of January 2018)